A General Type for Storage Operators

Karim NOUR
LAMA - Equipe de Logique, Université de Chambéry
73376 Le Bourget du Lac
e-mail nour@univ-savoie.fr

Abstract

In 1990, J.L. Krivine introduced the notion of storage operator to simulate, in λ-calculus, the "call by value" in a context of a "call by name". J.L. Krivine has shown that, using Gődel translation from classical into intuitionistic logic, we can find a simple type for storage operators in $AF2$ type system.
In this present paper, we give a general type for storage operators in a slight extension of $AF2$. We give at the end (without proof) a generalization of this result to other types.

1 Introduction

In 1990, J.L. Krivine introduced the notion of storage operators (see [3]). They are closed λ-terms which allow, for a given data type (the type of integers, for example), to simulate in λ-calculus the "call by value" in a context of a "call by name" (the head reduction).

J.L. Krivine has shown that the formula $\forall x\{N^*[x] \to \neg\neg N[x]\}$ is a specification for storage operators for Church integers : where $N[x]$ is the type of integers in second order logic, and the operation $*$ is the simple Gődel translation from classical into intuitionistic logic which associates to every formula F the formula F^* obtained by replacing in F each atomic formula with its negation (see [3]).

Some authors have been interested in the research of a most general type for storage operators. For example, V. Danos and L. Regnier have given as type for storage operators the formula $\forall x\{N^e[x] \to \neg\neg N[x]\}$ where the operation e is an elaborate Gődel translation which associates to every formula F the formula F^e obtained by replacing in F each

atomic formula $X(\overline{t})$ by $X_1(\overline{t}), ..., X_r(\overline{t}) \rightarrow \perp$ (see [1]). J.L. Krivine and the author have given a more general type for storage operators : the formula $\forall x\{N^g[x] \rightarrow \neg\neg N[x]\}$ where the operation g is the general Gődel translation which associates to every formula F the formula F^g obtained by replacing in F each atomic formula $X(\overline{t})$ by a formula $G_X[\overline{t}/\overline{x}]$ ending with $\perp$ (see [4] and [5]).

With the types cited before, we cannot type the following simple storage operators :
$T = \lambda\nu\lambda f((\nu)(T_i)\nu f)\lambda x x$ and $T' = \lambda\nu\lambda f((\nu)(T_i)\nu f)\lambda d(T_j)\nu f$ where T_i ($i = 1$ or 2) are the standard storage operators for integers (see [3]). This is due to the fact that the normal form of T (and T') contains a variable ν applied to two arguments and another ν applied to three arguments. Therefore, we cannot type T and T' because the variable ν is assigned by $N^*[x]$ (for example) and thus the number of the ν-arguments is fixed once for all.

To solve the problem, we will replace $N^*[x]$ in the type of storage operators by another type $N^\perp[x]$ which does not limit the number of ν-arguments and only enables to generate formulas ending with $\perp$ in order to find a general specification for storage operators.

The specifications of storage operators that we have obtained up to now do not explain that these operators only accept integers (for example $\lambda n\lambda f\lambda z(x)z$ is a normal λ-term of type $N^*[0]$). We will see that the type $N^\perp[x]$ is also a specification for the integers.

In this paper, we give a general type for the storage operators for integers in a slight extension of $AF2$ (the storage operators T and T' are typable of this type). We give at the end (without proof) a generalization of this result to the $\forall$-positive types (the universal second order quantifier appears positively in these types).

Acknowledgement. We wish to thank J.L. Krivine for helpful discussions. He found independently the principal result of this paper which he proved by a semantical method.

2 Definitions and notations

2.1 The pure λ-calculus

Let $t, u, u_1, ..., u_n$ be λ-terms, the application of t to u is denoted by $(t)u$. In the same way we write $(t)u_1...u_n$ instead of $(...((t)u_1)...)u_n$.
The β-equivalence relation is denoted by $u \simeq_\beta v$.
The notation $\sigma(t)$ represents the result of the simultaneous substitution σ to the free variables of t after a suitable renaming of the bounded variables of t.
We denote by $(u)^n v$ the λ-term $(u)...(u)v$ where u occurs n times, and $\overline{u}$ the sequence of λ-terms $u_1, ..., u_n$. If $\overline{u} = u_1, ..., u_n$, we denote by $(t)\overline{u}$ the λ-term $(t)u_1...u_n$.
Let us recall that a λ-term t either has a head redex [i.e. $t = \lambda x_1...\lambda x_n(\lambda xu)vv_1...v_m$, the head redex being $(\lambda xu)v$], or is in head normal form [i.e. $t = \lambda x_1...\lambda x_n(x)v_1...v_m$].
The notation $u \succ v$ means that v is obtained from u by some head reductions.
If $u \succ v$, we denote by $h(u, v)$ the length of the head reduction between u and v.
A λ-term t is said solvable iff the head reduction of t terminates.

Lemma 2.1 *(see [3]) If $u \succ v$, then :*
1) for every substitution σ, $\sigma(u) \succ \sigma(v)$ and $h(\sigma(u), \sigma(v)) = h(u, v)$.
2) for every sequence of λ-terms $\overline{w}$, there is a w such that $(u)\overline{w} \succ w$, $(v)\overline{w} \succ w$, and $h((u)\overline{w}, w) = h((v)\overline{w}, w) + h(u, v)$.

2.2 The $AF2$ type system

The types will be formulas of second order predicate logic over a given language.
The logical connectives are $\bot$ (for absurd), $\rightarrow$, and $\forall$.
There are individual (or first order) variables denoted by $x, y, z, ...$, and predicate (or second order) variables denoted by $X, Y, Z, ...$
We do not suppose that the language has a special constant for equality. Instead, we define the formula $u = v$ (where u, v are terms) to be $\forall Y(Y(u) \rightarrow Y(v))$ where Y is a unary predicate variable. Such a formula will be called an equation.
The formula $F_1 \rightarrow (F_2 \rightarrow (... \rightarrow (F_n \rightarrow G)...))$ is also denoted by $F_1, F_2, ..., F_n \rightarrow G$.
For every formula A, we denoted by $\neg A$ the formula $A \rightarrow \bot$.
If $\overline{v} = v_1, ..., v_n$ is a sequence of variables, we denoted by $\forall \overline{v} A$ the formula $\forall v_1...\forall v_n A$.
Let t be a λ-term, A a type, and $\Gamma = x_1 : A_1, ..., x_n : A_n$ a context. We define by the mean of this following rules the notion " t is of type A in the context Γ ". This notion is denoted by $\Gamma \vdash t : A$.

(1) $\Gamma \vdash x_i : A_i \ 1 \leq i \leq n$.

(2) If $\Gamma, x : A \vdash t : B$, then $\Gamma \vdash \lambda x t : A \to B$.

(3) If $\Gamma \vdash u : A \to B$, and $\Gamma \vdash v : A$, then $\Gamma \vdash (u)v : B$.

(4) If $\Gamma \vdash t : A$, then $\Gamma \vdash t : \forall x A$. (*)

(5) If $\Gamma \vdash t : \forall x A$, then $\Gamma \vdash t : A[u/x]$. (**)

(6) If $\Gamma \vdash t : A$, then $\Gamma \vdash t : \forall X A$. (*)

(7) If $\Gamma \vdash t : \forall X A$, then $\Gamma \vdash t : A[G/X]$. (**)

(8) If $\Gamma \vdash t : A[u/x]$, then $\Gamma \vdash t : A[v/x]$. (***)

The previous rules are subject to the following restrictions :
(*) The variable x (resp. X) has no free occurence in Γ.
(**) u is a term and G is a formula of the language.
(***) u and v are terms such that $u = v$ is a consequence of a given set of equations.

This type λ-calculus system is called $AF2$ (for arithmétique fonctionnelle du second ordre).

Theorem 2.1 *(see [2]) The $AF2$ type system has the following properties :*
1) Type is preserved during reduction.
2) Typable λ-terms are strongly normalizable.

We define on the set of types the two binary relations $\lhd$ and $\approx$ as the least reflexive and transitive binary relations such that :

- $\forall x A \lhd A[u/x]$, if u is a term of language ;

- $\forall X A \lhd A[F/X]$, if F is a formula of language ;

- $A \approx B$ iff $A = C[u/x]$, $B = C[v/x]$, and $u = v$ is a consequence of a given set of equations.

Theorem 2.2 *(see [5] and [7])*
1) Let A be an atomic formula. If $\Gamma \vdash t : A$, then t does not begin by λ.
2) If $\Gamma, x : A \vdash (x)u_1 ... u_n : B$, then :
$n = 0$, $A \lhd C$, $C \approx C'$, $B = \forall \overline{v} C'$, and $\overline{v}$ have no free occurence in Γ and A,
or

$n \geq 1$, $A \lhd C_1 \to B_1$, $B'_i \lhd C_{i+1} \to B_{i+1}$ $1 \leq i \leq n-1$, $B'_n \lhd B_{n+1}$, $B = \forall \overline{v} B'_{n+1}$ *where* $B_i \approx B'_i$ $1 \leq i \leq n+1$, $\Gamma, x : A \vdash u_i : C_i$ $1 \leq i \leq n$, *and* $\overline{v}$ *have no free occurence in* Γ *and* A.

3 The Church integers

Each data type can be defined by a second order formula. For example, the type of integers is the formula :

$$N[x] = \forall X\{X(0), \forall y(X(y) \to X(sy)) \to X(x)\}$$

where X is a unary predicate variable, 0 is a constant symbol for zero, and s is a unary function symbol for successor.
The formula $N[x]$ means semantically that x is an integer iff x belongs to each set X containing 0 and closed under the successor function s.
The λ-term $\underline{0} = \lambda x \lambda f x$ is of type $N[0]$ and represents zero.
The λ-term $\underline{s} = \lambda n \lambda x \lambda f (f)((n)x)f$ is of type $\forall y(N[y] \to N[s(y)])$ and represents the successor function.
A set of equations E is said adequate with the type of integers iff :

- $s(a) = 0$ is not an equational consequence of E ;

- If $s(a) = s(b)$ is an equational consequence of E, then so is $a = b$.

In the rest of the paper, we assume that all the set of equations are adequate with the type of integers.

For each integer n, we define the Church integer $\underline{n}$ by $\underline{n} = \lambda x \lambda f (f)^n x$.

Theorem 3.1 *(see [2]) For each integer* n, $\underline{n}$ *is the unique normal* λ*-term of type* $N[s^n(0)]$.

The propositional trace

$$N = \forall X\{X, (X \to X) \to X\}$$

of $N[x]$ also defines the integers.

Theorem 3.2 *(see [2]) A normal* λ*-term is of type* N *iff it is of the form* $\underline{n}$, *for a certain integer* n.

Remark A very important property of data type is the following (we express it for the type of integers) : in order to get a program for a function $f : N \to N$ it is sufficient to prove $\vdash \forall x(N[x] \to N[f(x)])$. For example a proof of $\vdash \forall x(N[x] \to N[p(x)])$ from the equations $p(0) = 0$, $p(s(x)) = x$ gives a λ-term for the predecessor in Church intergers (see [2]). $\Box$

4 The storage operators

Let T be a closed λ-term. We say that T is a storage operator for the integers iff for every $n \geq 0$, there is $\tau_n \simeq_\beta \underline{n}$, such that for every $\theta_n \simeq_\beta \underline{n}$, there is a substitution σ, such that $(T)\theta_n f \succ (f)\sigma(\tau_n)$.

Remark Let F be any λ-term (for a function), and θ_n a λ-term β-equivalent to $\underline{n}$. During the computation of $(F)\theta_n$, θ_n may be computed each time it comes in head position. Instead of computing $(F)\theta_n$, let us look at the head reduction of $(T)\theta_n F$. Since it is $\{(T)\theta_n f\}[F/f]$, by Lemma 2.1, we shall first reduce $(T)\theta_n f$ to its head normal form, which is $(f)\sigma(\tau_n)$, and then compute $(F)\sigma'(\tau_n)$. The computation has been decomposed into two parts, the first being independent of F. This first part is essentially a computation of θ_n, the result being τ_n, which is a kind of normal form of θ_n. The substitutions made in τ_n have no computational significance, since $\underline{n}$ is closed. So, in the computation of $(T)\theta_n F$, θ_n is computed first, and the result is given to F as an argument, T has stored the result, before giving it, as many times as needed, to any function. $\Box$

Examples If we take :
$T_1 = \lambda n((n)\delta)G$ where $G = \lambda x\lambda y(x)\lambda z(y)(\underline{s})z$ and $\delta = \lambda f(f)\underline{0}$
$T_2 = \lambda n\lambda f(((n)f)F)\underline{0}$ where $F = \lambda x\lambda y(x)(\underline{s})y$,
then it is easy to check that : for every $\theta_n \simeq_\beta \underline{n}$, $(T_i)\theta_n f \succ (f)(\underline{s})^n\underline{0}$ $(i = 1$ or $2)$.
Therefore T_1 and T_2 are two storage operators for the integers. $\Box$

It is a remarkable fact that we can give simple types to storage operators for integers. We first define the simple Gődel translation F^* of a formula F : it is obtained by replacing in the formula F, each atomic formula A by $\neg A$. For example :

$$N^*[x] = \forall X\{\neg X(0), \forall y(\neg X(y) \to \neg X(sy)) \to \neg X(x)\}$$

It is well know that, if F is provable in classical logic, then F^* is provable in intuitionistic logic.

We can check that $\vdash T_1, T_2 : \forall x\{N^*[x] \to \neg\neg N[x]\}$. And, in general, we have the following Theorem :

Theorem 4.1 *(see [3] and [6]) If* $\vdash T : \forall x\{N^*[x] \to \neg\neg N[x]\}$, *then* T *is a storage operator for the integers.*

Remark Let $\theta_0 = \lambda x\lambda f\lambda z(x)(\lambda dz)\lambda xx$.
It is easy to check that $\vdash \theta_0 : N^*[0]$, and $(T_2)\theta_0 f \succ (f)(\lambda d\underline{0})\lambda xx$.
Therefore T_2 is not a storage operator for the set $\{t \ / \vdash t : N^*[s^n(0)]\ n \geq 0\}$. □

The previous definition is not well adapted to study the storage operators. Indeed, it is, a priori, a Π_4^0 statement $(\forall n\exists\tau_n\forall\theta_n\exists\sigma A(T, n, \tau_n, \theta_n, \sigma))$. We will show (Theorem 4.2) that it is in fact equivalent to a Π_1^0 statement (τ_n can be computed from n, and σ from θ_n).

Let ν and f two fixed variables.
We denoted by $x_{n,a,b,\overline{c}}$ (where n is an integer, a, b two λ-terms, and $\overline{c}$ a finite sequence of λ-terms) a variable which does not appear in $a, b, \overline{c}$.

Theorem 4.2 *(see [5] and [8]) A closed* λ*-term* T *is a storage operators for the integers iff for every* $n \geq 0$, *there is a finite sequence of head reduction* $\{U_i \succ V_i\}_{1\leq i\leq r}$ *such that :*
1) $U_1 = (T)\nu f$ *and* $V_r = (f)\tau_n$ *where* $\tau_n \simeq_\beta \underline{n}$ *;*
2) $V_i = (\nu)ab\overline{c}$ *or* $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $0 \leq l \leq n-1$*;*
3) If $V_i = (\nu)ab\overline{c}$, *then* $U_{i+1} = (a)\overline{c}$ *if* $n = 0$ *and* $U_{i+1} = ((b)x_{n-1,a,b,\overline{c}})\overline{c}$ *if* $n \neq 0$ *;*
4) If $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $0 \leq l \leq n-1$, *then* $U_{i+1} = (a)\overline{d}$ *if* $l = 0$ *and* $U_{i+1} = ((b)x_{l-1,a,b,\overline{d}})\overline{d}$ *if* $l \neq 0$.

5 General type for storage operators

5.1 The $AF2_\perp$ type system

In this section, we present a slight extension of the $AF2$ type system denoted by $AF2_\perp$.

We assume that for every integer n, there is a countable set of special n-ary second order variables denoted by $X_\perp, Y_\perp, Z_\perp$...., and called $\perp$-variables.

A type A is called an $\perp$-type iff A is obtained by the following rules :

- $\bot$ is an $\bot$-type ;

- $X_\bot(t_1, ..., t_n)$ is an $\bot$-type ;

- If B is an $\bot$-type, then $A \to B$ is an $\bot$-type for every type A ;

- If A is an $\bot$-type, then $\forall vA$ is an $\bot$-type for every variable v.

Therefore, A is an $\bot$-type iff : $A = \forall \overline{v_1}(E_1 \to F_1)$, $F_i = \forall \overline{v_{i+1}}(E_{i+1} \to F_{i+1})$ $1 \leq i \leq r-1$, and $F_r = \forall \overline{v_{r+1}} X_\bot(t_1, ..., t_n)$ or $F_r = \forall \overline{v_{r1}}\ \bot$.

We add to the $AF2$ type system the new following rules :

(6′) If $\Gamma \vdash t : A$, and $X_\bot$ has no free occurence in Γ, then $\Gamma \vdash t : \forall X_\bot A$.

(7′) If $\Gamma \vdash t : \forall X_\bot A$, and G is an $\bot$-type, then $\Gamma \vdash t : A[G/X_\bot]$.

We call $AF2_\bot$ the new type system, and we write $\Gamma \vdash_\bot t : A$ if t is typable in $AF2_\bot$ of type A in the context Γ.

Remark We can also see the system $AF2_\bot$ as a restriction of the system $AF2$. Therefore, $AF2_\bot$ satisfies the same properties of $AF2$ (strongly normalization and preservation of types). □

5.2 The general Theorem

Let

$$N^\bot[x] = \forall X_\bot\{X_\bot(0), \forall y(X_\bot(y) \to X_\bot(sy)) \to X_\bot(x)\}$$

where $X_\bot$ is a unary $\bot$-variable.
By the previous remark, we have : if $\Gamma \vdash_\bot t : N^\bot[s^n(0)]$, then $t \simeq_\beta \underline{n}$.

Lemma 5.1 *If T is a closed normal λ-term such that $\vdash T : \forall x\{N^*[x] \to \neg\neg N[x]\}$, then $\vdash_\bot T : \forall x\{N^\bot[x] \to \neg\neg N[x]\}$.*

Proof T is a closed normal λ-term, then $T = \lambda\nu T'$, and $\nu : N^*[x] \vdash T' : \neg\neg N[x]$. Since $\nu : N^\bot[x] \vdash_\bot \nu : N^*[x]$, then $\nu : N^\bot[x] \vdash_\bot T' : \neg\neg N[x]$. Therefore $\vdash_\bot T : \forall x\{N^\bot[x] \to \neg\neg N[x]\}$. □

Remarks
1) We have $\vdash T_1, T_2 : \forall x\{N^\bot[x] \to \neg\neg N[x]\}$.
2) The λ-terms T and T' (given in the introduction) are of type $\forall x\{N^\bot[x] \to \neg\neg N[x]\}$.

- We have $\nu : N^{\perp}[x] \vdash_{\perp} \nu : \perp, (\perp \rightarrow \perp) \rightarrow \perp$. Since $\nu : N^{\perp}[x], f : \neg N[x] \vdash_{\perp} (T_i)\nu f : \perp$ and $\vdash_{\perp} \lambda x x : \perp \rightarrow \perp$, then $\nu : N^{\perp}[x], f : \neg N[x] \vdash_{\perp} ((\nu)(T_i)\nu f)\lambda x x : \perp$. Therefore $\vdash_{\perp} T : \forall x\{N^{\perp}[x] \rightarrow \neg\neg N[x]\}$.

- We have $\nu : N^{\perp}[x] \vdash_{\perp} \nu : \perp, (\perp \rightarrow \perp) \rightarrow \perp$. Since $\nu : N^{\perp}[x], f : \neg N[x] \vdash_{\perp} (T_i)\nu f : \perp$ and $\nu : N^{\perp}[x], f : \neg N[x] \vdash_{\perp} \lambda d(T_i)\nu f : \perp \rightarrow \perp$, then $\nu : N^{\perp}[x], f : \neg N[x] \vdash_{\perp} ((\nu)(T_i)\nu f)\lambda d(T_i)\nu f : \perp$. Therefore $\vdash_{\perp} T' : \forall x\{N^{\perp}[x] \rightarrow \neg\neg N[x]\}$. $\Box$

We give now a general type for storage operators for integers.

Theorem 5.1 *If $\vdash_{\perp} T : \forall x\{N^{\perp}[x] \rightarrow \neg\neg N[x]\}$, then T is a storage operator for the integers.*

The type system $F_{\perp}$ is the subsystem of $AF2_{\perp}$ where we only have propositional variables and constants (predicate variables or predicate symbols of arity 0). So, first order variable, function symbols, and finite sets of equations are useless. The rules for typed are 1), 2), 3), and 6), 7) restricted to propositional variables. For each predicate variable (resp. predicate symbol) X, we associate a predicate variable (resp. a predicate symbol) $X^{\diamond}$ of $F_{\perp}$ type system. For each formula A of $AF2_{\perp}$, we associate the formula $A^{\diamond}$ of $F_{\perp}$ obtained by forgetting in A the first order part. If $\Gamma = x_1 : A_1, ..., x_n : A_n$ is a context of $AF2_{\perp}$, then we denote by $\Gamma^{\diamond}$ the context $x_1 : A_1^{\diamond}, ..., x_n : A_n^{\diamond}$ of $F_{\perp}$.
We write $\Gamma \vdash_{\perp}^{\diamond} t : A$ if t is typable in $F_{\perp}$ of type A in the context Γ.
We have obviously the following property : if $\Gamma \vdash_{\perp} t : A$, then $\Gamma^{\diamond} \vdash_{\perp}^{\diamond} t : A^{\diamond}$.

Theorem 5.1 is a consequence of the following Theorem.

Theorem 5.2 *If $\vdash_{\perp}^{\diamond} T : N^{\perp} \rightarrow \neg\neg N$, then for every $n \geq 0$, there is an $m \geq 0$ and $\tau_m \simeq_{\beta} \underline{m}$, such that for every $\theta_n \simeq_{\beta} \underline{n}$, there is a substitution σ, such that $(T)\theta_n f \succ (f)\sigma(\tau_m)$.*

Indeed, if $\vdash_{\perp} T : \forall x\{N^{\perp}[x] \rightarrow \neg\neg N[x]\}$, then $\vdash_{\perp}^{\diamond} T : N^{\perp} \rightarrow \neg\neg N$. Therefore for every $n \geq 0$, there is an $m \geq 0$ and $\tau_m \simeq_{\beta} \underline{m}$, such that for every $\theta_n \simeq_{\beta} \underline{n}$, there is a substitution σ, such that $(T)\theta_n f \succ (f)\sigma(\tau_m)$. We have $\vdash_{\perp} \underline{n} : N[s^n(0)]$, then $f : \neg N[s^n(0)] \vdash_{\perp} (T)\underline{n} f : \perp$, therefore $f : \neg N[s^n(0)] \vdash_{\perp} (f)\underline{m} : \perp$. By Theorem 2.2, we have $\vdash_{\perp} \underline{m} : N[s^n(0)]$ and thus $n = m$. Therefore T is a storage operator for the integers. $\Box$

In order to prove Theorem 5.2, we shall need some Lemmas.

Lemma 5.2 *If* $\Gamma, \nu : N^{\perp} \vdash^{\diamond}_{\perp} (\nu)\overline{d} :\perp$, *then* $\overline{d} = a, b, d_1, ..., d_r$ *and there is an* $\perp$*-type* F, *such that :* $\Gamma, \nu : N^{\perp} \vdash^{\diamond}_{\perp} a : F$; $\Gamma, \nu : N^{\perp} \vdash^{\diamond}_{\perp} b : F \to F$; $F \lhd E_1 \to F_1$, $F_i \lhd E_{i+1} \to F_{i+1}$ $1 \leq i \leq r-1$; $F_r \lhd \perp$; *and* $\Gamma, \nu : N^{\perp} \vdash^{\diamond}_{\perp} c_i : E_i$ $1 \leq i \leq r$.

Proof We use Theorem 2.2. □

Lemma 5.3 *If* F *is an* $\perp$*-type and* $\Gamma, x : F \vdash^{\diamond}_{\perp} (x)\overline{d} :\perp$, *then* $\overline{d} = d_1, ..., d_r$; $F \lhd E_1 \to F_1$; $F_i \lhd E_{i+1} \to F_{i+1}$ $1 \leq i \leq r-1$; $F_r \lhd \perp$; *and* $\Gamma, x : F \vdash^{\diamond}_{\perp} c_i : E_i$ $1 \leq i \leq r$.

Proof We use Theorem 2.2. □

Lemma 5.4 *Let* t *be a normal* λ*-term, and* $A_1, ..., A_n$ *a sequence of* $\perp$*-types.*
If $x_1 : A_1, ..., x_n : A_n \vdash^{\diamond}_{\perp} t : N$, *then there is an* $m \geq 0$ *such that* $t = \underline{m}$.

Proof We prove by induction on u that if u is a normal λ-term, X a propositionnal variable, and $x_1 : A_1, ..., x_n : A_n, x : X, f : X \to X \vdash^{\diamond}_{\perp} u : X$, then there is an $m \geq 0$ such that $u = (f)^m x$. □

We can now give the proof of Theorem 5.2.

Proof of Theorem 5.2
Let ν and f two fixed variables, and $\vdash^{\diamond}_{\perp} T : N^{\perp} \to \neg\neg N$.
A good context Γ is a context of the form $\Gamma = \nu : N^{\perp}, f : \neg N, x_{n_1,a_1,b_1,\overline{c_1}} : F_1, ..., x_{n_p,a_p,b_p,\overline{c_p}} : F_p$ where F_i is an $\perp$-type, and $\Gamma \vdash^{\diamond}_{\perp} a_i : F_i$, $\Gamma \vdash^{\diamond}_{\perp} b_i : F_i \to F_i$, $0 \leq n_i \leq n-1$, and $1 \leq i \leq p$.

We will prove that for every $n \geq 0$, there is a finite sequence of head reduction $\{U_i \succ V_i\}_{1 \leq i \leq r}$ such that :
1) $U_1 = (T)\nu f$ and $V_r = (f)\tau$ where $\tau \simeq_{\beta} \underline{m}$ for some $m \geq 0$;
2) $V_i = (\nu)ab\overline{c}$ or $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $0 \leq l \leq n-1$;
3) If $V_i = (\nu)ab\overline{c}$, then $U_{i+1} = (a)\overline{c}$ if $n = 0$ and $U_{i+1} = ((b)x_{n-1,a,b,\overline{c}})\overline{c}$ if $n \neq 0$
4) If $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $0 \leq l \leq n-1$, then $U_{i+1} = (a)\overline{d}$ if $l = 0$ and $U_{i+1} = ((b)x_{l-1,a,b,\overline{d}})\overline{d}$ if $l \neq 0$.
5) There is a good context Γ such that $\Gamma \vdash^{\diamond}_{\perp} V_i :\perp$ $1 \leq i \leq r$.

We have $\vdash^{\diamond}_{\perp} T : N^{\perp} \to \neg\neg N$, then $\nu : N^{\perp}, f : \neg N \vdash^{\diamond}_{\perp} (T)\nu f :\perp$, and by Lemmas 5.2 and 5.3, $(T)\nu f \succ V_1$ where $V_1 = (f)\tau$ or $V_1 = (\nu)ab\overline{c}$.
Assume that we have the head reduction $U_k \succ V_k$ and $V_k \neq (f)\tau$.

- If $V_k = (\nu)ab\overline{c}$, then, by induction hypothesis, there is a good context Γ such that $\Gamma \vdash^{\diamond}_{\perp} (\nu)ab\overline{c} : \perp$. By Lemma 5.2, there is an $\perp$-type, such that $\Gamma \vdash^{\diamond}_{\perp} a : F$; $\Gamma \vdash^{\diamond}_{\perp} b : F \to F$; $\overline{c} = c_1, ..., c_s$; $F \lhd E_1 \to F_1$; $F_i \lhd E_{i+1} \to F_{i+1}$ $1 \leq i \leq s-1$; $F_s \lhd \perp$; and $\Gamma \vdash^{\diamond}_{\perp} c_i : E_i$ $1 \leq i \leq s$.

  - If $n = 0$, let $U_{k+1} = (a)\overline{c}$. We have $\Gamma \vdash^{\diamond}_{\perp} U_{k+1} : \perp$.

  - If $n \neq 0$, let $U_{k+1} = ((b)x_{n-1,a,b,\overline{c}})\overline{c}$. The variable $x_{n-1,a,b,\overline{c}}$ is not used before. Indeed, if it is, we check easly that the λ-term $(T)\underline{n}f$ is not solvable. But that is impossible because $f : \neg N \vdash^{\diamond}_{\perp} (T)\underline{n}f : \perp$. Let $\Gamma' = \Gamma, x_{n-1,a,b,\overline{c}} : F$. Γ' is a good context and $\Gamma' \vdash^{\diamond}_{\perp} U_{k+1} : \perp$.

- If $V_k = (x_{l,a,b,\overline{c}})\overline{d}$, then, by induction hypothesis, there is a good context Γ such that $\Gamma \vdash^{\diamond}_{\perp} (x_{l,a,b,\overline{c}})\overline{d} : \perp$. $x_{l,a,b,\overline{c}} : F$ is in the context Γ, then by Lemma 5.3, $\overline{d} = d_1, ..., d_s$; $F \lhd E_1 \to F_1$; $F_i \lhd E_{i+1} \to F_{i+1}$ $1 \leq i \leq s-1$; $F_s \lhd \perp$; and $\Gamma \vdash^{\diamond}_{\perp} d_i : E_i$ $1 \leq i \leq s$.

  - If $l = 0$, let $U_{k+1} = (a)\overline{d}$. We have $\Gamma \vdash^{\diamond}_{\perp} U_{k+1} : \perp$.

  - If $l \neq 0$, let $U_{k+1} = ((b)x_{l-1,a,b,\overline{d}})\overline{d}$. The variable $x_{l-1,a,b,\overline{d}}$ is not used before. Indeed, if it is, we check easly that the λ-term $(T)\underline{n}f$ is not solvable. But that is impossible because $f : \neg N \vdash^{\diamond}_{\perp} (T)\underline{n}f : \perp$. Let $\Gamma' = \Gamma, x_{l-1,a,b,\overline{c}} : F$. Γ' is a good context and $\Gamma' \vdash^{\diamond}_{\perp} U_{k+1} : \perp$.

Therefore there is a good context Γ' such that $\Gamma' \vdash^{\diamond}_{\perp} U_{k+1} : \perp$, then, by Lemmas 5.2 and 5.3, $U_{k+1} \succ V_{k+1}$ where $V_{k+1} = (f)\tau$ or $V_{k+1} = (\nu)ab\overline{c}$ or $V_{k+1} = (x_{l,a,b,\overline{c}})\overline{d}$ $0 \leq l \leq n-1$. This constraction always terminates. Indeed, if not, we check easly that the λ-term $(T)\underline{n}f$ is not solvable. But that is impossible because $f : \neg N \vdash^{\diamond}_{\perp} (T)\underline{n}f : \perp$.
Therefore there is $r \geq 0$ and a good context Γ such that $\Gamma \vdash^{\diamond}_{\perp} V_r = (f)\tau : \perp$, and by Theorem 2.2, $\Gamma \vdash^{\diamond}_{\perp} \tau : N$. Therefore by Lemma 5.4, there is an $m \geq 0$ such that $\tau \simeq_\beta \underline{m}$. By the Theorem 4.2, we have the proof of the Theorem 5.2. □

6 Generalization

In this section, we give (without proof) a generalization of the Theorem 5.1.

Let T be a closed λ-term, and D, E two closed types of $AF2$ type system. We say that T is a storage operator for the pair of types (D, E) iff for every λ-term $\vdash t : D$, there is λ-terms τ_t and τ'_t, such that $\tau'_t \simeq_\beta \tau_t$, $\vdash \tau_t : E$, and for every $\theta_t \simeq_\beta t$, there is a substitution

σ, such that $(T)\theta_t f \succ (f)\sigma(\tau_t)$.

We define two sets of types of $AF2$ type system: Ω^+ (set of $\forall$-positive types), and Ω^- (set of $\forall$-negative types) in the following way :

- If A is an atomic type, then $A \in \Omega^+$, and $A \in \Omega^-$;

- If $T \in \Omega^+$, and $T' \in \Omega^-$, then, $T' \to T \in \Omega^+$, and $T \to T' \in \Omega^-$;

- If $T \in \Omega^+$, then $\forall x T \in \Omega^+$;

- If $T \in \Omega^-$, then $\forall x T \in \Omega^-$;

- If $T \in \Omega^+$, then $\forall X T \in \Omega^+$;

- If $T \in \Omega^-$, and X has no free occurence in T, then $\forall X T \in \Omega^-$.

Therefore, T is a $\forall$-positive types iff the universal second order quantifier appears positively in T.

For each predicate variable X, we associate an $\perp$- variable $X_\perp$.
For each formula A of $AF2$ type system, we define the formula $A^\perp$ as follows :

- If $A = \perp$, then $A^\perp = A$;

- If $A = R(t_1, ..., t_n)$, where R is an n-ary predicate symbol, then $A^\perp = A$;

- If $A = X(t_1, ..., t_n)$, where X is an n-ary predicate variable, then $A^\perp = X_\perp(t_1, ..., t_n)$;

- If $A = B \to C$, then $A^\perp = B^\perp \to C^\perp$;

- If $A = \forall x B$, then $A^\perp = \forall x B^\perp$;

- If $A = \forall X B$, then $A^\perp = \forall X_\perp B^\perp$.

$A^\perp$ is called the $\perp$-transformation of A.

Theorem 6.1 *Let D, E two $\forall$-positive closed types of $AF2$ type system, such that E does not contain $\perp$. If $\vdash_\perp T : D^\perp \to \neg\neg E$, then T is a storage operator for the pair (D, E).*

References

[1] V. Danos and L. Regnier *Notes sur les opérateurs de mise en mémoire*
Manuscript, 1992

[2] J.L. Krivine *Lambda-calcul, types et modèles*
Masson, Paris 1990

[3] J.L. Krivine *Opérateurs de mise en mémoire et traduction de Gődel*
Archiv for Mathematical Logic 30, 1990, pp. 241-267

[4] J.L. Krivine *Mise en mémoire (preuve générale)*
Manuscript, 1993

[5] K. Nour *Opérateurs de mise en mémoire en lambda-calcul pur et typé*
Thèse de Doctorat, Université de Chambéry, 1993

[6] K. Nour *Une preuve syntaxique d'un théorème de J.L. Krivine sur les opérateurs de mise en mémoire*
CRAS Paris, t. 318, Série I, p. 201-204, 1994.

[7] K. Nour *Opérateurs de mise en mémoire et types $\forall$-positifs*
Manuscript, 1993

[8] K. Nour and R. David *Storage operators and directed λ-calculus*
To appear in J.S.L.